“THEME ARTICLE”, “FEATURE ARTICLE”, or “COLUMN” goes here: The theme topic or column/department name goes after the colon.

 DOI 10.1109/MCSE.2019.2918766

Heterogeneous Hierarchical Workflow Composition

Orcun Yildiz
Argonne National Laboratory

Jorge Ejarque
Barcelona Supercomputing Center

Henry Chan
Argonne National Laboratory

Subramanian Sankaranarayanan
Argonne National Laboratory

Rosa M. Badia
Barcelona Supercomputing Center

Spanish National Research Council

Tom Peterka
Argonne National Laboratory

Workflow systems promise scientists an automated end-to-end path from hypothesis to discovery. However, expecting any single workflow system to deliver such a wide range of capabilities is impractical. A more practical solution is to compose the end-to-end workflow from more than one system. With this goal in mind, the integration of task-based and in situ workflows is explored, where the result is a hierarchical heterogeneous workflow composed of subworkflows, with different levels of the hierarchy using different programming, execution, and data models. Materials science use cases demonstrate the advantages of such heterogeneous hierarchical workflow composition.

Scientific computing consists of multiple related computational tasks. For instance, the detection of highly turbulent potentially destructive features in plasma physics simulations requires the in situ coupling of two simulation codes, a compression tool, a feature detection algorithm, and a visualization library.[1] Such complex workflows require significant effort from scientists to manage the scheduling and data exchange among those tasks. To help automate this process, scientific workflow frameworks allow scientists to define the dependencies and data exchanges among connected tasks instead of managing those manually, potentially resulting in increased scientific productivity.

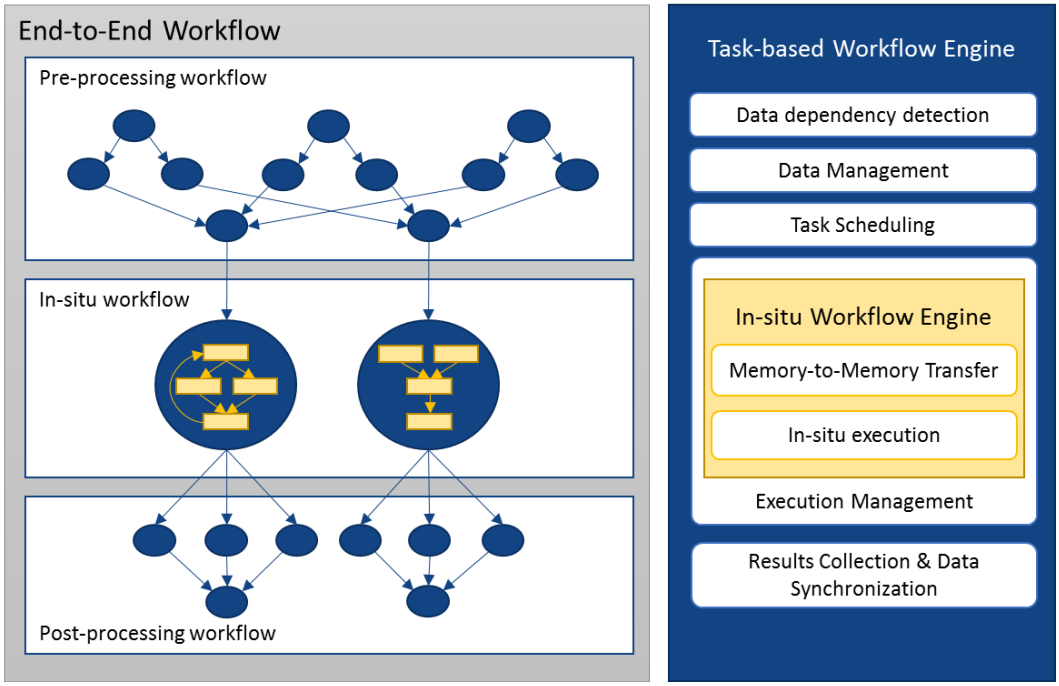


Figure 1. Overview of the proposed approach where in situ workflows are integrated as tasks in an end-to-end task-based workflow. The result is a heterogeneous workflow with two levels of hierarchy shown in blue and yellow colors using different programming, execution, and data models.

In this work, we consider two main classes of scientific workflows: in situ and task-based. In situ workflows run within a single high-performance computing (HPC) system, and the data exchange is done through memory or the supercomputer interconnect during the same scheduled execution of a job.[2-5] Task-based workflows are task-parallel, high-throughput workflows that exchange data through files, can run across several independent systems in a wide area such as grids and clouds but do not have to.[6-9] Heterogeneous workflow systems such as in situ and task-based ones lack interoperability with each other. Hence, today's workflow solutions often cannot support an automated end-to-end path from hypothesis to discovery.

With the goal of automating the process of scientific discovery, we investigate the integration of task-based and in situ HPC workflows. Figure 1 shows an example of our approach, where we employ entire in situ workflows as single tasks of a high-level end-to-end workflow that is managed by a task-based workflow tool. The result is a heterogeneous two-level hierarchy that is designed to integrate two completely different workflow programming paradigms: task-based, high-throughput workflows with in situ HPC workflows. This integration requires providing (1) an interface to describe in situ computations in a task-based workflow without adding extra complexity and (2) efficient execution of the in situ computations without interfering with the other tasks of the task-based workflow.

We evaluated our approach by performing experiments in the field of materials science. In particular, the in situ workflow consists of a molecular dynamics simulation coupled with a parallel in situ feature detector that selects nucleated molecule clusters during crystallization. Meanwhile, the task-based workflow launches an ensemble of in situ workflows with different initial conditions iterating until nucleation succeeds, saves results from experiments that successfully nucleated, and collects snapshots from those ensemble members into an animation.

Our results reveal that heterogeneous workflow integration can benefit both task-based and in situ workflow systems. We believe that the insights and lessons learned from our integration will increase understanding and motivate further research into heterogeneous workflow composition.

In Situ Workflows

In situ workflows address the mismatch between I/O and computing capabilities of supercomputers by allowing users to perform data analysis at simulation time. This in turn minimizes the I/O time and shortens the overall time to discovery.

Several in situ workflow solutions have been proposed. One example is Decaf[5], middleware for building and executing in situ workflows, allowing parallel communication of coupled tasks by creating communication channels over HPC interconnects through MPI. It provides a Python API to describe the workflow graph. Decaf does not impose any constraints on this graph topology and can manage graphs with cycles.

Some widely used visualization tools also extended their APIs to expose the simulation data in situ to their visualization engines. Examples of this approach include VisIt's Libsim[2] and ParaView's Catalyst[3] libraries.

We have also witnessed the extension of I/O libraries to support in situ analysis. For example, ADIOS,[11] originally designed for I/O staging, can now couple analysis and visualization applications together with simulations through an I/O interface.[4] Another example is Damaris[12], I/O middleware that supports in situ data processing and visualization using dedicated cores or nodes of the simulation platform.

Distributed Workflows

Workflow management systems for distributed computing became popular a decade ago as a tool to deploy scientific workflows in grid computing environments. Several survey papers[13-14] present taxonomies of representative systems from that period.

Distributed workflow systems execute in different ways. For example, Taverna[6] orchestrates previously deployed web services. The web service model offers some advantages in decoupled environments and fits well in cloud computing, but its performance overheads are not well suited for HPC workflows. Most workflow systems are oriented to compose external binaries and tools (e.g., Galaxy[7]), and some can directly invoke methods described in programming languages (e.g., PyCOMPSs/COMPSs[8] and Swift[10]).

The use of parallelism inside a task of a distributed workflow has been limited and until recently, most distributed workflow systems managed serial tasks. In some cases, tasks invoke parallel MPI applications. For example, the fusion community use Kepler[15] as a workflow management system. Another example is the climate community, who use tailored workflow management systems based on tagged scripts (Cylc[16] or Autosubmit[17]); their workflows typically comprise several invocations of simulations, programmed with MPI, which are offloaded for execution to the job scheduler of a cluster or supercomputer.

Hybrid Workflows

End-to-end scientific applications can require different types of computation capabilities during their lifetime. For instance, the Cybershake earthquake hazard model contains MPI jobs that in turn generate distributed jobs that will run on grids.[18] For this purpose, Cybershake uses Pegasus[9] as a workflow management system. Our approach is similar, but rather than a single MPI program, we integrate entire HPC subworkflows of multiple interconnected MPI tasks (communicating with each other through the HPC interconnect using MPI) inside the task-based workflow. Next, we present one such heterogeneous workflow integration.

HETEROGENEOUS WORKFLOW INTEGRATION

Traditionally, scientists only executed a single simulation that could be easily managed with modifying a configuration file or running a simple shell script. Nowadays, the complexity of scientific computing has increased significantly, and scientists require the combination of different types of computations to perform their experiments. Tedious effort is required to automate all the

necessary steps to obtain scientific results. These steps include data management to transfer the required data among the workflow tasks, the configuration of the different simulations and workflow tools to perform the desired computations, and the efficient coordination of the different executions on the available resources. Managing all these steps can be burdensome for scientists unfamiliar with the complexity of parallel programming. Moreover, some dynamic workflows, such as our science use case that launches in situ tasks until a rare event is encountered, cannot be described with a static script.

Here, we present our methodology to hide the complexity by integrating task-based workflows with in situ workflows. With this integration, we can automate the end-to-end path from hypothesis to discovery by using these different workflows for what they do best. Our approach consists of employing one or more in situ subworkflows as tasks of a high-level end-to-end workflow, thereby integrating these in situ computations with the other pre/post processing phases. The end-to-end workflow is managed by a task-based workflow tool that performs data dependency analysis, schedules the tasks on available resources, performs the required data transfers, manages the task execution, and retrieves the results of computations. In situ subworkflows can be described inside this task-based workflow by indicating their input/output data and computational requirements. This approach is sufficient to enable the task-based workflow tool to manage entire in situ workflows as single tasks in terms of data dependencies, scheduling, and data transfers. For execution of these in situ subworkflows, the task-based workflow tool configures them to use the assigned resources and provides the location of the input and the output data. Next, the task-based workflow tool starts the execution of the in situ subworkflows and monitors them until completion. Then, the task-based workflow tool retrieves the results and makes them available to other tasks of the end-to-end workflow.

```
1   @constraint(ComputingUnits=num_cores)
2   @decaf(dfScript="decaf-config-script.py")
3   @task(infile=FILE_IN, outfile=FILE_OUT, returns=int)
4   def decaf_task_method(infile, outfile):
5     pass
6
7   @constraint(ComputingUnits=num_cores)
8   @task(outfile=FILE_OUT) #in_param input object as default
9   def normal_task(in_param, outfile):
10    ...
11  if __name__ == "__main__":
12    normal_task(param, file1)
13    decaf_task_method(file1, file2)
```

Figure 2. Decaf and normal task description and invocation in PyCOMPSs. Decaf tasks are described similarly to normal tasks.

We implemented our approach using PyCOMPSs and Decaf as the task-based and in situ workflow tools, respectively. PyCOMPSs enables users to program task-based parallel workflows as Python scripts by annotating Python methods with a *task* decorator. Inside this decorator, users annotate the direction of the method arguments as input or output. Besides the task decorator, users can specify constraints for a task (e.g., amount of memory, number of CPU cores) with the *constraint* decorator. With this information, the PyCOMPSs runtime automatically detects data dependencies between the annotated tasks and builds a directed acyclic graph (DAG), where nodes represent task executions, and edges represent data dependencies. If the PyCOMPSs code contains loop statements, PyCOMPSs will unroll these cycles and represent them as a DAG. We will see this occurring in our use case. According to this DAG and the defined constraints, task executions are scheduled and remotely executed on the available resources automatically. Decaf, on the other hand, is a parallel dataflow engine designed to perform efficient in situ computations over HPC interconnects through MPI. Decaf has a Python API, where users can define the different nodes and edges of a dataflow and the number of processes assigned to each dataflow

entity. Once the dataflow is defined, it is executed as a multiple-program-multiple-data (MPMD) MPI application.

To integrate a Decaf execution in a PyCOMPSs workflow, we have extended the PyCOMPSs syntax with a *decaf* decorator, which is used to identify and describe the task as an in situ Decaf subworkflow. Figure 2 shows an example of a simple usage of the *decaf* decorator. Users need to define a method without the implementation details, which has the same arguments as the Decaf execution. In the *task* and *constraint* decorators, users state the direction of the parameters and the resource requirements as for a normal PyCOMPSs task. In the *decaf* decorator, users indicate that the defined task is a Decaf subworkflow whose description is provided in the *dfScript* property. Once the Decaf task is defined, users can invoke it as a regular Python method call and combine with other PyCOMPSs tasks as shown in Figure 2 (lines 12 and 13).

When the PyCOMPSs runtime detects a Decaf invocation, it treats the entire subworkflow as a single PyCOMPSs task; analyzes the data dependencies of the new node in the DAG; and, once the task has satisfied its data dependencies, schedules it on the available resources according to the defined constraints. Then, PyCOMPSs transfers the required data to the selected computing nodes and issues the command to proceed with the Decaf execution. PyCOMPSs provides the number of MPI processes to spawn (the *ComputingUnits* specified in the *constraint* decorator), the MPI hostfile, and any command-line arguments, and generates the command to execute the workflow as a MPMD MPI execution. This command is automatically executed by the PyCOMPSs worker runtime, which monitors the execution state. Once the Decaf execution finishes, the PyCOMPSs master runtime is notified in order to proceed with the remaining tasks in the workflow graph that depend on the results of this Decaf execution.

Science Use Case

The materials science problem we study using our heterogeneous workflow composition is nucleation as a material cools and crystallizes. In particular, we study water freezing, but the same workflow applies to nucleation in many other material systems.

Application Description

Understanding the mechanisms and kinetics of crystallization events is key to understanding a wide range of natural and technological systems. Nucleation is a stochastic event that requires a large number of statistics to elucidate its kinetics. Capturing nucleation in simulations is difficult, especially during the early stages when only a few atoms have crystallized.

Scientists simulate nucleation by running either a single large simulation or many instances of small simulations. However, both scenarios have their drawbacks. For a large simulation, massive computing power is needed, prohibiting most scientists from conducting such computations. On the other hand, observing nucleation in small simulation systems is difficult. To increase their chances, scientists employ advanced sampling techniques by performing many instances of crystallization simulations. Given the stochastic nature of the nucleation process, nucleation can be observed in only a few of these instances. Managing such ensemble workflows can be tedious—especially without the aid of workflow systems—where scientists need to run and analyze many simulation instances hoping to achieve the results they need from only a few such instances. We demonstrate the applicability of our heterogeneous workflow composition for both scenarios: one large molecular system and many small systems.

Diamond Structure Classifier

The analysis of the molecular system requires identifying the different phases of ice—hexagonal, cubic, and amorphous/liquid—by using a diamond structure classifier originally implemented in the OVITO visualization software. Briefly, the classifier labels crystalline atoms based on the topological relationship between the nearby first shell and second shell neighbors. Unlabeled atoms, including atoms in the grain boundaries, are considered amorphous.

We developed a parallel version of the diamond structure classifier and coupled it in situ with the LAMMPS[19] molecular dynamics model using Decaf. Our in situ subworkflow consists of two parallel MPI HPC tasks—LAMMPS and our diamond classifier—communicating with each other using the HPC interconnect and MPI. This allowed us to avoid the potential I/O bottleneck associated with writing/reading all atoms to/from storage; instead we detected diamond-shaped structures in situ alongside LAMMPS. To further improve the I/O performance, we stored time steps only after a minimum number of atoms (1/8 of the total number of atoms) crystallized. Moreover, within a time step we stored only the crystallized atoms.

To find the optimal number of processes for the parallel diamond structure classifier, we performed a scalability test where we used a LAMMPS input file consisting of 2,000,000 atoms. We used up to 128 MPI processes. Optimal performance was achieved when using 16 processes. The reason is that each subdomain in the parallel decomposition includes the nearest first and second neighbors of any atom in the subdomain. As the subdomains shrink in size with increasing numbers of processes, the size of this ghost region dominates the computation, limiting scalability.

EXPERIMENTAL METHODOLOGY

Platform Description

Our experiments were conducted on the MareNostrum 4 (MN4) supercomputer, which has 3,456 computing nodes, each with 2 Intel Xeon Platinum 8160 CPUs with 24 cores each at 2.1 GHz, and 2 GB or 8 GB of DDR4 RAM memory per core. All nodes are connected to each other by a 100 Gb Intel Omni-Path interconnection network and a 10 Gb Ethernet network.

The experiments in MN4 were executed as a batch job using the PyCOMPSs submission scripts, which request a set of computing nodes. One of the assigned nodes acts as a PyCOMPSs master, and the rest of the nodes act as workers. PyCOMPSs manages the allocation and execution of the Decaf subworkflows in a subset of these worker nodes.

The MN4 supercomputer was the only facility that our science use case required. PyCOMPSs, as a distributed task-based workflow tool, supports the execution of workflow tasks in different computing environments as demonstrated in previous work.[20] However, distributing the computation among multiple sites is not the objective of our current work.

Experimental Deployment

We studied the nucleation problem with two simulation sizes: the large simulation is a molecular model of a box of water composed of 2,000,000 atoms, and the small simulation is composed of 4,000 atoms. For the large simulation, PyCOMPSs launched a single instance of a Decaf subworkflow, which used 2,000 processes for LAMMPS with a temperature value of 210K, and 16 processes for the diamond structure classifier. The results of the Decaf subworkflow were further post-processed and rendered by the remainder of the end-to-end workflow, which was also parallelized and managed by PyCOMPSs. For the small simulation, many instances of LAMMPS crystallization simulations were performed. In this regime, PyCOMPSs launched multiple instances of Decaf subworkflows with different configurations, and evaluated the results to check whether one of the simulation instances nucleated. This cycle was repeated until a nucleation event occurred or a limit on the number of cycles was reached. In this experiment, we used three different undercoolings (210K, 220K, and 230K) with different random seed values, each Decaf instance using 44 processes for LAMMPS and 4 processes for the parallel diamond structure classifier.

Figure 3 shows the hierarchical workflow graph for the ensemble of such simulations. The graph is generated from the perspective of the end-to-end task-based workflow, such that the in situ subworkflows appear as single nodes. Blue nodes are the *generate configurations* tasks that generate a configuration file with different initial conditions (seed variable, temperature) for each

simulation instance. Red nodes represent the Decaf subworkflows. White nodes are *get atoms* tasks that fetch the output files from these subworkflows. Pink nodes are *render* tasks, and the dark red node is the *generate animation* task, which combines the rendered frames into an animation.

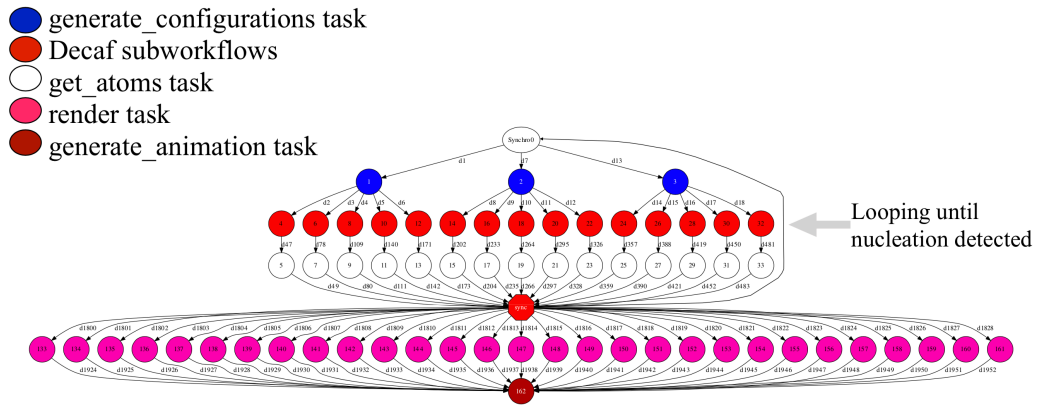


Figure 3. Hierarchical workflow graph dynamically executed by PyCOMPSs, which loops on the first set of tasks until a successful nucleation event occurs. Blue nodes generate a configuration file with different initial conditions (seed variable, temperature) for each simulation instance; red nodes represent the Decaf subworkflows; white nodes fetch the output files from these subworkflows and evaluate the results to check for nucleation; the red hexagon is the synchronization point to decide whether to run another cycle or to proceed with the post-processing phase, where pink nodes are render tasks and the dark red node combines the rendered frames into an animation.

The task graph is dynamic and varies depending on whether nucleation succeeds. When nucleation does not occur, PyCOMPSs discards the output files produced by Decaf subworkflows and continues spawning more of these subworkflows. When it detects a successful nucleation, the workflow executes the *render* and *generate animation* tasks to create the animation.

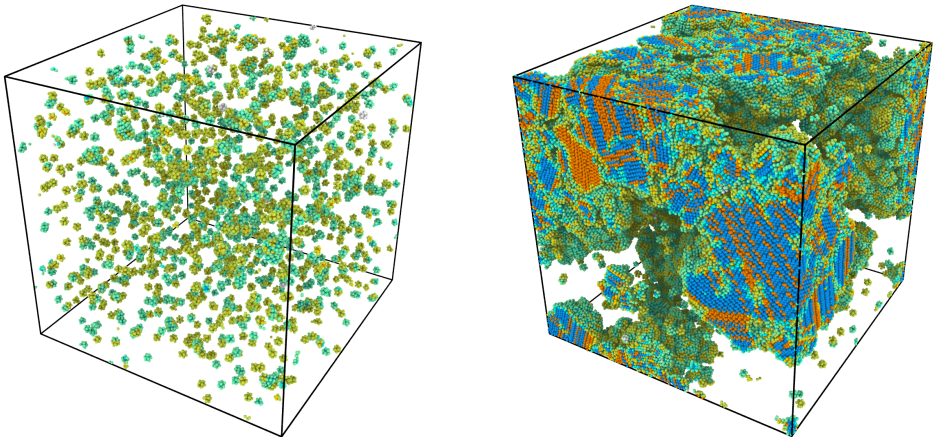
Figure 3 shows that we can compose dynamic and complex workflow graphs by using our approach. Moreover, our approach has widespread applicability to other domains. The scenario of running multiple simulation instances to sample rare events is common, for example, in protein folding, self-assembled structures, and genetic algorithms.

Capturing Nucleation in a Single Large Simulation Instance

We ran LAMMPS for 5,000,000 iterations and performed the diamond structure analysis every 10,000 iterations. Figure 4 shows the different phases of ice nucleation we were able to observe with our heterogeneous workflow model. This result demonstrates that our hierarchical workflow composition can support the full science pipeline from the hypothesis of a nucleation event to the animation of a successful nucleation without any human interaction. To succeed, we used each system for what it does best. The task-based workflow system combines different workflow tasks (generating configuration files for different simulation instances, launching in situ subworkflows, rendering) while the in situ workflow system accelerates the performance of this hierarchical workflow by performing the detection of diamond-shaped structures in situ alongside the molecular dynamics simulations.

Our approach has also several benefits compared with the traditional post hoc detection of diamond structures. The first is time savings, as shown in Table I. Normally, scientists have to perform the post-processing manually (i.e., diamond structure classification by using OVITO visualization software) for each of the 500 snapshots generated by the simulation. Our results indicate that this takes 12 seconds on average per snapshot, or a total of approximately 17 hours

of post-processing time. This is a conservative estimate that does not include the extra time required to transfer files to the post-processing environment, the time for reading files in OVITO, and the user interaction time. We hide this post-processing time by detecting diamond-shaped structures in situ alongside LAMMPS.



(a) before nucleation (b) nucleating

Figure 4. Stages of nucleation in freezing water system composed of 2,000,000 atoms

Table I. Time and storage requirements for the different approaches when simulating the nucleation process. With a hierarchical workflow, we can fully automate the science path from hypothesis to discovery without requiring any human interaction. This also results in 64x storage savings.

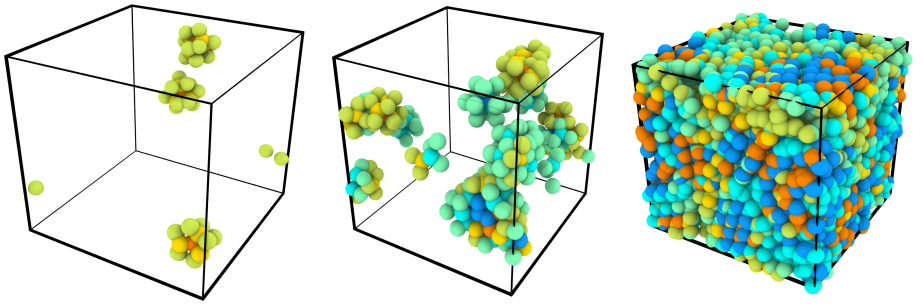
Approach	Simulation Time	Post-processing Time	End-to-End Execution Time	Storage Requirements
Heterogeneous workflow	11 hours	-	11 hours	250 MB
Traditional (post hoc)	11 hours	17+ hours	28+ hours	16 GB

The second benefit is space savings, as shown in Table I. We reduced the storage requirements by storing only the positions of the nucleated atoms in the LAMMPS output files and assembling those into an animation, which requires only 250 MB storage space. On the other hand, the traditional approach saves all of the 500 output files (32 MB each) generated by LAMMPS for further post-processing. This output occupies 16 GB of storage, which is 64 times higher than our approach does.

Although we observed nucleation with a single large simulation, this problem size requires massive computing power, prohibiting most scientists from conducting such computations. For example, this experiment used 22,780 core-hours on the MN4 supercomputer. Next, we use fewer water molecules, which is more difficult to nucleate.

Capturing Nucleation by Running Many Instances of Small Simulations

We launched 60 instances of a Decaf subworkflow at three undercoolings (210K, 220K, and 230K) with PyCOMPSs. We ran LAMMPS for 5,000,000 iterations and performed the diamond structure analysis every 10,000 iterations. We observed nucleation in only 1 of the 60 simulation instances with 210K because of the stochastic nature of the nucleation process. Moreover, capturing the nucleation with higher temperatures becomes much harder. We discarded the snapshots generated from unsuccessful simulation instances and saved only the results from experiments that successfully nucleated. In addition to saving 98% of storage space, this approach allows scientists to focus on the results that they need. Figure 5 shows the stages of this nucleation in a freezing water system composed of 4,000 atoms.



(a) before nucleation

(b) subcritical nuclei (starting to nucleate)

(c) nucleating

Figure 5. Stages of nucleation in freezing water system composed of 4,000 atoms

This result further demonstrates that the heterogeneous workflow model supports the automated end-to-end science path for a complex science problem, which would otherwise require significant effort from scientists. The dynamic nature of the task-based workflow, which afforded flexible control of running multiple simulation instances, and in situ coupling of the diamond detector with LAMMPS using the in situ subworkflow, were the keys to being able to capture rare events in an ensemble of simulations.

Table II. Number of processes and core-hours used by each simulation problem on MareNostrum. With hierarchical workflow, we can simulate the nucleation process by using 94% fewer computing resources.

Simulation Size	Processes	Wall-Clock Time	Core-Hours
Large (2M)	2,016	11 hours	22,780
Small (4K)	480	2.5 hours	1,210

Moreover, by using much fewer resources, our hierarchical workflow composition achieved results similar to those with large simulations. For instance, this experiment used only 1,210 core-hours on the MN4 supercomputer compared with the large simulation problem's use of 22,780 core-hours (94% reduction in the required resources), as shown in Table II. This result shows that our approach can help scientists who usually do not have an access to large computing resources conduct such experiments.

CONCLUSION

Our results reveal that heterogeneous workflow integration is advantageous to both task-based and in situ workflow systems. On the one hand, we can accelerate the performance of task-based workflows by providing access to HPC dataflow, where information is exchanged through shared memory or HPC interconnect instead of files. On the other hand, this composition of workflows extends in situ workflows with dynamicity and different programming models such as bag-of-tasks and looping that are not available in the dataflow HPC programming model. Our solution also combines Python and C++ workflow tasks, which can lead to the easier implementation of multilevel workflows for science applications.

Several avenues remain open for future work. One is to expand our scientific use cases to include distributed multifacility environments. Currently, we are exploring a particle accelerator physics problem that involves simulations running on HPC systems, experiments on particle accelerators, and optimization algorithms on workstations. In the short term, we will also investigate extending our system to support finer-grained synchronization. Coarse-grained synchronization model is common in most workflow tools where tasks in the workflow graph can start their execution only after the tasks on which they depend finish their execution. With finer-grained synchronization, one could start rendering the individual output files as soon as they are produced by the in situ subworkflows instead of waiting for an entire simulation snapshot. Our long-term goal is to create a common interface for different workflow systems to communicate with each other. A standardized interface would allow the propagation of the required information for workflow composition (e.g., resource requirements, input and output file locations, executable information) without being specific to any one workflow tool.

ACKNOWLEDGMENTS

This work is a collaboration between Argonne National Laboratory and the Barcelona Supercomputing Center within the Joint Laboratory for Extreme-Scale Computing. This research is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC02-06CH11357, program manager Laura Biven, and by the Spanish Government (SEV2015-0493), by the Spanish Ministry of Science and Innovation (contract TIN2015-65316-P), by Generalitat de Catalunya (contract 2014-SGR-1051).

REFERENCES

1. J. Y. Choi *et al.*, “Coupling exascale multiphysics applications: Methods and lessons learned,” in *Proceedings of IEEE International Conference on eScience*, 2018.
2. T. Kuhlen *et al.*, “Parallel in situ coupling of simulation with a fully featured visualization system,” in *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV)*, 2011.
3. U. Ayachit *et al.*, “Paraview catalyst: Enabling in situ data analysis and visualization,” in *Proceedings of the First*

Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization. ACM, pp. 25–29, 2015.

4. F. Zheng *et al.*, "PreDatA – preparatory data analytics on peta-scale machines," in *Proceedings of the IEEE International Symposium on Parallel Distributed Processing (IPDPS '10)*, 2010.
5. M. Dreher and T. Peterka, "Decaf: Decoupled dataflows for in situ high-performance workflows," Argonne National Laboratory, Lemont, IL, Tech. Rep. ANL/MCS-TM-371, 2017.
6. D.Hull *et al.*, "Taverna: A tool for building and running workflows of services," *Nucleic Acids Research*, vol. 34, no. suppl. 2, pp. W729–W732, 2006.
7. E. Afgan *et al.*, "The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update," *Nucleic Acids Research*, p. gkw343, 2016.
8. E. Tejedor *et al.*, "Pycompss: Parallel computational workflows in python," *The International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 66–82, 2017.
9. E. Deelman *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.
10. M. Wilde et al., "Swift: A language for distributed parallel scripting," *Parallel Computing,* vol. 37, no. 9, pp. 633–652, 2011.
11. J. F. Lofstead *et al.*, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," in *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments*, ser. CLADE '08. New York, NY: ACM, 2008.
12. M. Dorier *et al.*, "Damaris: Addressing performance variability in data management for post-petascale simulations," *ACM Transactions on Parallel Computing (TOPC)*, vol. 3, no. 3, p. 15, 2016.
13. J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 171–200, 2005.
14. E. Deelman *et al.*, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.
15. I. Altintas *et al.*, "Kepler: An extensible system for design and execution of scientific workflows," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management,* IEEE, pp. 423–424, 2004.

16. H. J. Oliver et al., (2018). Cylc: A workflow engine for cycling systems. Journal of Open Source Software, vol. 3, no. 27, p. 737, 2018.
17. D. Manubens-Gil *et al.*, "Seamless management of ensemble climate prediction experiments on HPC platforms," in *International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, pp. 895–900, 2016.
18. R. Graves *et al.*, "Cybershake: A physics-based seismic hazard model for southern California," *Pure and Applied Geophysics*, vol. 168, no. 3–4, pp. 367–381, 2011.
19. S. Plimpton *et al.*, "LAMMPS - large-scale atomic/molecular massively parallel simulator," *Sandia National Laboratories*, vol. 18, p. 43, 2007.
20. E. Tejedor, J. Álvarez, and R. M. Badia. "Infrastructure-agnostic programming and interoperable execution in heterogeneous grids," *Computing and Informatics*, vol. 35, no. 4, pp. 986–1004, 2017.

ABOUT THE AUTHORS

Orcun Yildiz is a postdoctoral researcher in the Mathematics and Computer Science Division at Argonne National Laboratory. He received his Ph.D. degree in computer science from Ecole Normale Superieure de Rennes (France) in December 2017. His research interests include scientific workflows, big data processing, I/O management, and high-performance computing. Contact him at oyildiz@anl.gov.

Jorge Ejarque holds a Ph.D. from the Technical University of Catalonia (UPC) in 2015. He is a researcher at the Barcelona Supercomputing Center contributing to the design and development of tools and programming models for distributed computing. He has worked in several national and international R&D projects and was a member of the experts' board of the Spanish National Grid Initiative. His current research interests are parallel programming models for distributed platforms and energy-efficient execution of distributed applications. Contact him at jorge.ejarque@bsc.es.

Henry Chan is a postdoctoral researcher at the Center for Nanoscale Materials (CNM), Argonne National Laboratory. He received his Ph.D. in computational chemistry from the University of Illinois at Chicago in 2015. He has extensive experience in simulations of soft-matter material systems and has published in high-impact scientific journals. His current research focus is on the application of machine learning principles in the development of interatomic force fields

and techniques that enable accurate large-scale molecular simulations. Contact him at hchan@anl.gov.

Subramanian Sankaranarayanan is a scientist in the Nanoscale Science and Technology (NST) Division at Argonne and a senior scientist at the Institute of Molecular Engineering at the University of Chicago. He obtained his Ph.D. in chemical engineering from the University of South Florida in 2007. Prior to joining Argonne, Subramanian was a postdoctoral fellow at the School of Engineering and Applied Sciences at Harvard University from 2007 to 2010. His research at Argonne focuses on the use of machine learning to bridge the electronic, atomistic and mesoscopic scales. He uses supervised machine learning techniques to develop first-principles-based force fields for simulating reactive and mesoscopic systems. Other interests include machine learnt computational tool development for integrated X-ray imaging of ultrafast energy transport across solid-solid and solid-liquid interfaces. His interests span a diverse range of applications from tribology and corrosion to neuromorphic computing and thermal management. Contact him at ssankaranarayanan@anl.gov.

Rosa M. Badia holds a Ph.D. in computer science (1994) from the Technical University of Catalonia (UPC). She is a researcher from the Spanish National Research Council (CSIC) and team leader of the Workflows and Distributed Computing research group at the Barcelona Supercomputing Center. She was involved in teaching and research activities at UPC from 1989 to 2008, where she was an associate professor from 1997. Her current research interests are programming models for complex platforms (from multicore CPUs, to GPUs, to grid and cloud systems). She has published more than 150 papers in international conferences and journals and has participated in several international R&D projects. Contact her at rosa.m.badia@bsc.es.

Tom Peterka is a computer scientist in the Mathematics and Computer Science Division at Argonne National Laboratory, scientist at the University of Chicago Consortium for Advanced Science and Engineering, adjunct assistant professor at the University of Illinois at Chicago, and fellow of the Northwestern Argonne Institute for Science and Engineering. His research interests are large-scale parallel in situ analysis of scientific data. He received his Ph.D. in computer science from the University of Illinois at Chicago in 2007, and he currently works actively in several DOE- and NSF-funded projects. Contact him at tpeterka@mcs.anl.gov.